```go
package main

// Запуск из командной строки:  go test -bench . 4squares_test.go

import (
    "testing"
    "math"
//  “fmt”
)

func search2rec(sum int, amount int, result []int) {
    if amount == 0  {
        if sum == 0 {
//            fmt.Println(result)
        }
        return
    }
    var start int
    if len(result) == 0 {
        start = 0
    } else {
        start = result[len(result)-1]
    }
    for i:= start; i*i <= sum; i++ {
        search2rec(sum - i*i, amount - 1, append(result, i))
    }
}

func search2cycle(sum int) {
    for i1:= 0; i1*i1 <= sum; i1++ {
        for i2:= i1; i2*i2 <= sum; i2++ {
            for i3:= i2; i3*i3 <= sum; i3++ {
                for i4:= i3; i4*i4 <= sum; i4++ {
                    if i1*i1 + i2*i2 + i3*i3 + i4*i4 == sum {
//                      fmt.Println(i1, i2, i3, i4)
                    }
                }
            }
        }
    }
}

func search3rec(sum int, amount int, result []int) {
    if amount == 0  {
        if sum == 0 {
//            fmt.Println(result)
        }
        return
    }
    var start int
    if len(result) == 0 {
        start = 0
    } else {
        start = result[len(result)-1]
    }
    for i:= start; i*i*amount <= sum; i++ {
        search3rec(sum - i*i, amount - 1, append(result, i))
    }
}

```

```
func search3cycle(sum int) {
    for i1:= 0; i1*i1*4 <= sum; i1++ {
        for i2:= i1; i2*i2*3 <= sum - i1*i1; i2++ {
            for i3:= i2; i3*i3*2 <= sum - i1*i1 - i2*i2; i3++ {
                for i4:= i3; i4*i4 <= sum - i1*i1 - i2*i2 - i3*i3; i4++ {
                    if i1*i1 + i2*i2 + i3*i3 + i4*i4 == sum {
//                      fmt.Println(i1, i2, i3, i4)
                    }
                }
            }
        }
    }
}

func PerfectSquare (n int) (sqrt int, is bool) {
    sqrt = int( math.Round( math.Sqrt( float64(n) ) ) )
    return sqrt, sqrt*sqrt==n
}

func search4rec(sum int, amount int, result []int) {
    if amount == 1  {
        if _, ok:= PerfectSquare(sum); ok {
//          fmt.Println(append(result, x))
        }
        return
    }
    var start int
    if len(result) == 0 {
        start = 0
    } else {
        start = result[len(result)-1]
    }
    for x:= start; x*x*amount <= sum; x++ {
        search4rec(sum - x*x, amount - 1, append(result, x))
    }
}

func search4cycle(sum int) {
    for x1:= 0; x1*x1*4 <= sum; x1++ {
        for x2:= x1; x2*x2*3 <= sum - x1*x1; x2++ {
            for x3:= x2; x3*x3*2 <= sum - x1*x1 - x2*x2; x3++ {
                if _, ok:= PerfectSquare(sum - x1*x1 - x2*x2 - x3*x3); ok {
//                      fmt.Println(x1, x2, x3, x4)
                }
            }
        }
    }
}

// Названия тестируемых функций должны начинаться на Benchmark,
// за которым идёт название, начинающееся с большой буквы

func BenchmarkSearch2Recursive(b *testing.B) {
    for i := 0; i < b.N; i++ {
        search2rec(50, 4, make([]int, 0))
    }
}

func BenchmarkSearch2Cycle(b *testing.B) {
    for i := 0; i < b.N; i++ {
        search2cycle(50)
    }
}
```

```
124
125 func BenchmarkSearch3Recursive(b *testing.B) {
126     for i := 0; i < b.N; i++ {
127         search3rec(50, 4, make([]int, 0))
128     }
129 }
130
131 func BenchmarkSearch3Cycle(b *testing.B) {
132     for i := 0; i < b.N; i++ {
133         search3cycle(50)
134     }
135 }
136
137 func BenchmarkSearch4Recursive(b *testing.B) {
138     for i := 0; i < b.N; i++ {
139         search4rec(50, 4, make([]int, 0))
140     }
141 }
142
143 func BenchmarkSearch4Cycle(b *testing.B) {
144     for i := 0; i < b.N; i++ {
145         search4cycle(50)
146     }
147 }
```

4squares_test.go

```
1  package main
2
3  import "fmt"
4
5  func solve (bricks []int, rest int, solution []int) {
6      if rest == 0 {
7          fmt.Println(solution)
8          return
9      }
10     if len(bricks) == 0 {
11         return
12     }
13     // либо мы берём первый кирпич, ...
14     if rest >= bricks[0] {
15         solve(bricks[1:], rest - bricks[0], append(solution, bricks[0]))
16     }
17     // ... либо не берём
18     solve (bricks[1:], rest, solution)
19 }
20
21 func main() {
22     bricks:= []int{7, 11, 24, 3, 28, 4, 6, 12}
23     carrying:= 48
24     solve(bricks, carrying, make([]int, 0))
25 }
```

backpack.go

```go
package main

import "fmt"

const N = 5

type queen struct {
              col int
              row int
           }

func abs(x int) int {
    if x >= 0  {
        return x
    } else {
        return -x
    }
}

func Connected (q1, q2 queen) bool {
    return q1.col == q2.col ||
           q1.row == q2.row ||
           abs(q1.col - q2.col) == abs (q1.row - q2.row)
}

func Conflict (qs []queen, q queen) bool {
    for _, q2 := range qs {
        if Connected (q, q2) { return true }
    }
    return false
}

func main() {
    var Queens [N]queen
    for col0:= 0; col0 < N; col0++ {
        Queens[0] = queen{col0, 0}
        if Conflict(Queens[:0], Queens[0]) { continue }
        for col1:= 0; col1 < N; col1++ {
            Queens[1] = queen{col1, 1}
            if Conflict(Queens[:1], Queens[1]) { continue }
            for col2:= 0; col2 < N; col2++ {
                Queens[2] = queen{col2, 2}
                if Conflict(Queens[:2], Queens[2]) { continue }
                for col3:= 0; col3 < N; col3++ {
                    Queens[3] = queen{col3, 3}
                    if Conflict(Queens[:3], Queens[3]) { continue }
                    for col4:= 0; col4 < N; col4++ {
                        Queens[4] = queen{col4, 4}
                        if Conflict(Queens[:4], Queens[4]) { continue }
                        for _, q := range Queens {
                            fmt.Printf("%c%d ", q.col+'a', q.row + 1)
                        }
                        fmt.Println()
                    }
                }
            }
        }
    }
}
```

8queens_cyclic.go (5 ферзей на самом деле)

```go
package main

import "fmt"

const N = 8

type queen struct {
              col int
              row int
           }

func abs(x int) int {
    if x >= 0  {
        return x
    } else {
        return -x
    }
}

func Connected (q1, q2 queen) bool {
    return q1.col == q2.col ||
           q1.row == q2.row ||
           abs(q1.col - q2.col) == abs (q1.row - q2.row)
}

func Conflict (qs []queen, q queen) bool {
    for _, q2 := range qs {
        if Connected (q, q2) { return true }
    }
    return false
}

var Queens [N]queen

func Search(n int) {
    if n == 0  {
        for _, q := range Queens {
            fmt.Printf("%c%d ", q.col+'a', q.row + 1)
        }
        fmt.Println()
        return
    }
    for col:= 0; col < N; col++ {
        Queens[N-n] = queen{col, N-n}
        if Conflict(Queens[:N-n], Queens[N-n]) { continue }
        Search(n-1)
    }
}

func main() {
    Search(N)
}
```

8queens_recursive.go

```go
package main

import "fmt"

var N, K int

var P []int //глобальный "массив"

func Cikl(CurrentPos int, FirstItem int) {
    // CurrentPos - степень вложенности цикла,
    // FirstItem - стартовое число очередного цикла
    for i := FirstItem; i <= N-(K-CurrentPos); i++ {
        P[CurrentPos-1] = i
        // Цикл вложенности K - последний.
        if CurrentPos == K {
            // Печатаем комбинацию (сочетание)
            fmt.Println(P)
        } else {
            Cikl(CurrentPos+1, i+1)
        }
    }
}

func main() {
    for {
        fmt.Print("Enter N: ")
        fmt.Scanln(&N)
        fmt.Print("Enter K: ")
        fmt.Scanln(&K)
        if K>0 && N>=K { break }
    }
    // Алгоритм: реализуем цикл вложенности K
    // for i1:= 1; i <= N-(K-1) {
    //     for i2:= i1+1; i <=  N-(K-2) {
    //         for i3:= i2+1; i <=  N-(K-3) {
    //           . . .      . . .      . . .
    //             for iK:= i(K-1)+1; To N-(K-K) {
    //                 Print (i1, i2, ..., iK)
    //             }
    //           . . .      . . .      . . .
    //         }
    //     }
    // }
    P = make([]int, K, K)
    Cikl(1, 1)
}
```

combinations.go

```
package main

import "fmt"

const n = 8

type
    cell struct {
        row, col int
    }

func (c cell) Connected(c2 cell) bool {
    // Соединены ли клетки c и c2 ходом ферзя?
    if c == c2 { return false }
    return    (c.row == c2.row) || (c.col == c2.col) ||
            (abs(c.row-c2.row) == abs(c.col-c2.col))
}

func (c cell) Print() {
    fmt.Printf("%c%d ", c.col+'a', c.row+1)
}

func (c cell) Next() cell {
    // Возвращает клетку, следующую за клеткой c.
    // Направление движения: вдоль столбца - увеличиваем строку,
    // в конце столбца переходим на нижнюю клетку следующего столбца
    if c.row < n-1  {
        return cell{c.row + 1, c.col}
    } else {
        return cell{0, c.col + 1}
    }
}

func (c cell) Terminal() bool {
    // Верно ли, что c - последняя клетка на доске?
    return c.Next().col == n
}

func Success(list []cell) bool {
    // Верно ли, что все клетки доски находятся
    // под боем какого-то ферзя из списка list
    for row:= 0; row < n; row++ {
        for col:= 0; col < n; col++ {
            ok := false
            for _, c:= range(list) {
                if (cell{row, col}).Connected(c) {
                    ok = true
                    break
                }
            }
            if !ok {
                return false
            }
        }
    }
    return true
}

```

```
var result []cell  // здесь храним текущее наилучшее решение

func search(list []cell) {
    if list[0].col == n-1 && list[0].row == n - len(list) {
    // терминальный случай: последняя комбинация ферзей,
    // дальше двигаться некуда
        return
    }
    if len(list) >= len(result)-1 {
    // добавлять ферзей бессмысленно - улучшить результат не удастся
        return
    }
    // last - последий ферзь ф текущем списке
    last:= list[len(list) - 1]
    // добавляем ещё одного ферзя
    for c:= last.Next(); !c.Terminal(); c = c.Next() {
        if Success( append(list, c) ) {
        // если новый ферзь делает список таким, что все
        // все клетки находятся под боем, то этот список
        // улучшает текущий результат - запоминаем его
            result = append(list, c)
            return
        }
        // если новый ферзь не далает список таким,
        // что все клетки находятся под боем,
        // то пытаемся добавить ещё ферзей
        search( append(list, c)    )
    }
}

func main() {
    // Начальное решение - заполняем ферзями весь нижний ряд
    for i:= 0; i< n; i++ {
        result = append(result, cell{0, i} )
    }
    // Поиск начинается со списка из одного ферзя,
    // стоящего в первой клетке - клетке {0, 0}
    search([]cell{cell{0,0}})
    // Печать результата
    for _, c := range (result) {
        c.Print()
    }
    fmt.Println()
}

func abs(x int) int {
    if x<0 {
        return -x
    } else {
        return x
    }
}
```

aggressive_queens.go